11 Using a relational database service: RDS

This chapter covers

- Launching and initializing relational databases with RDS
- Creating and restoring database snapshots
- Setting up a highly available database
- Tweaking database performance
- Monitoring databases

Relational databases are the de facto standard for storing and querying structured data, and many applications are built on top of a relational database system such as MySQL. Typically, relational databases focus on data consistency and guarantee ACID database transactions (atomicity, consistency, isolation, and durability). A typical task is storing and querying structured data like the accounts and transactions in an accounting application.

If you want to use a relational database on AWS, you have two options:

- Use the managed relational database service Amazon RDS, which is offered by AWS.
- Operate a relational database yourself on top of virtual machines.

The Amazon Relational Database Service (Amazon RDS) offers ready-to-use relational databases such as PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server. If your application supports one of these relational database systems, the migration to Amazon RDS is easy.

Beyond that, AWS offers its own engine called Amazon Aurora, which is MySQL- and PostgreSQL-compatible. If your application supports MySQL or PostgreSQL, the migration to Amazon Aurora is easy.

RDS is a managed service. The managed service provider—in this case AWS—is responsible for providing a defined set of services—in this case, operating a relational database system. Table 11.1 compares using an RDS database and hosting a database yourself on virtual machines.

Table 11.1 Managed service RDS vs. a self-hosted database on virtual machines

	Amazon RDS	Self-hosted on virtual machines
Cost for AWS services	Higher because RDS costs more than virtual machines (EC2)	Lower because virtual machines (EC2) are cheaper than RDS
Total cost of ownership	Lower because operating costs are split among many customers	Much higher because you need your own manpower to manage your database
Quality	AWS professionals are responsible for the managed service.	You'll need to build a team of professionals and implement quality control yourself.
Flexibility	High, because you can choose a relational database system and most of the configuration parameters	Higher, because you can control every part of the relational database system you installed on virtual machines

You'd need considerable time and know-how to build a comparable relational database environment based on virtual machines, so we recommend using Amazon RDS for relational databases whenever possible to decrease operational costs and improve quality. That's why we won't cover hosting your own relational database on VMs in this book. Instead, we'll introduce Amazon RDS in detail.

In this chapter, you'll launch a MySQL database with the help of Amazon RDS. Chapter 2 introduced a WordPress setup like the one shown in figure 11.1; you'll reuse this example in this chapter, focusing on the database part. After the MySQL database is up and running, you'll learn how to import, back up, and restore data. More advanced topics like setting up a highly available database and improving the performance of the database will follow.

Examples are 100% covered by the Free Tier

The examples in this chapter are totally covered by the Free Tier. As long as you don't run the examples longer than a few days, you won't pay anything for it. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days, because you'll clean up your account at the end of the chapter.

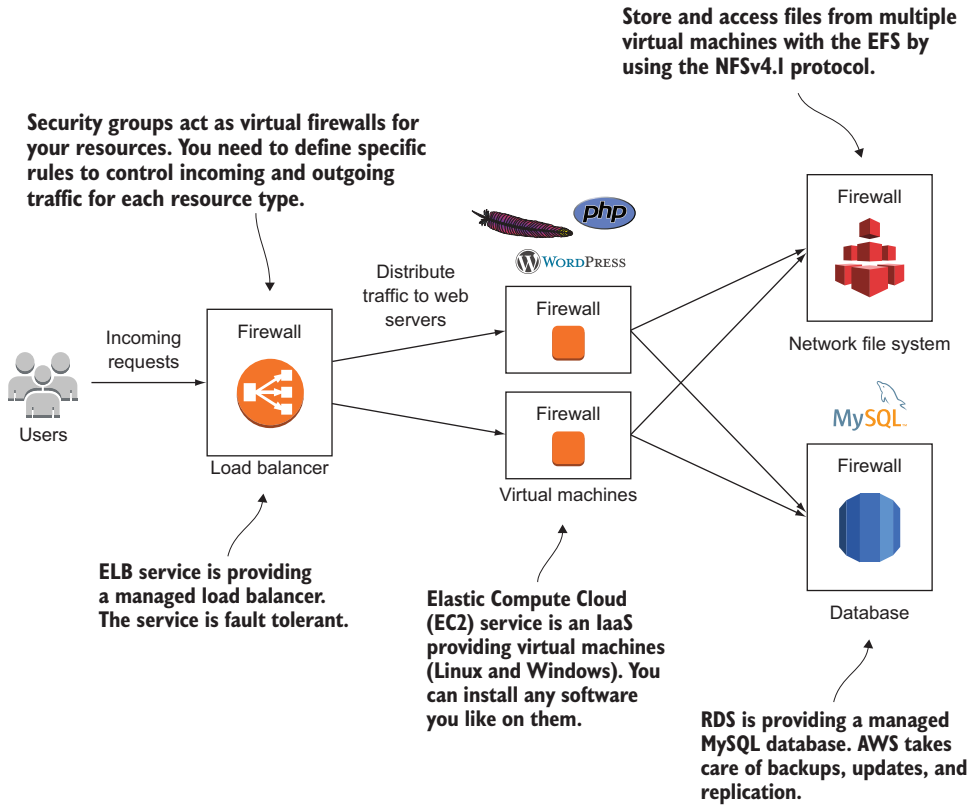

Figure 11.1 The company's blogging infrastructure consists of two load-balanced web servers running WordPress and a MySQL database server.

All the examples in this chapter use a MySQL database used by a WordPress application. You can easily transfer what you learn to other database engines such as Aurora, PostgreSQL, MariaDB, Oracle Database, Microsoft SQL Server, and to applications other than WordPress.

11.1 Starting a MySQL database

The popular blogging platform WordPress is built on top of a MySQL relational database. If you want to host a WordPress blog on your VM, you'll need to run the PHP application—for example, with the help of an Apache web server—and you'll need to operate a MySQL database where WordPress stores the articles, comments, and author accounts. Amazon RDS offers a MySQL database as a managed service, so you no longer need to install, configure, and operate a MySQL database yourself.

11.1.1 Launching a WordPress platform with an RDS database

Launching a database consists of two steps:

1. Launching a database instance
2. Connecting an application to the database endpoint

To set up a WordPress blogging platform with a MySQL database, you'll use the same CloudFormation template you used in chapter 2. You also used Amazon RDS there. The template can be found on GitHub and on S3. You can download a snapshot of the repository at https://github.com/AWSinAction/code2/archive/master.zip. The file we're talking about is located at chapter11/template.yaml. On S3, the same file is located at http://mng.bz/a4C8.

Execute the following command to create a CloudFormation stack containing an RDS database instance with a MySQL engine and web servers serving the WordPress application:

```
$ aws cloudformation create-stack --stack-name wordpress --template-url \
➥ https://s3.amazonaws.com/awsinaction-code2/chapter11/template.yaml \
➥ --parameters ParameterKey=KeyName,ParameterValue=mykey \
➥ ParameterKey=AdminPassword,ParameterValue=test1234 \
➥ ParameterKey=AdminEMail,ParameterValue=your@mail.com
```

You'll wait several minutes while the CloudFormation stack is created in the background, which means you'll have enough time to learn the details of the RDS database instance while the template is launching. Listing 11.1 shows parts of the CloudFormation template used to create the `wordpress` stack. Table 11.2 shows the attributes you need when starting an RDS database using CloudFormation or the Management Console.

Table 11.2 Attributes needed to connect to an RDS database

Attribute	Description
`AllocatedStorage`	Storage size of your database in GB
`DBInstanceClass`	Size (also known as instance type) of the underlying virtual machine
`Engine`	Database engine (Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, or Microsoft SQL Server) you want to use
`DBName`	Identifier for the database
`MasterUsername`	Name for the master user
`MasterUserPassword`	Password for the master user

It is possible to mark an RDS instance as publicly accessible. But we generally do not recommend you enable access from the internet to your database, to prevent unwanted

access. Instead, as shown in our example, an RDS instance should only be accessible within the VPC.

To connect to an RDS instance, you need an EC2 instance running in the same VPC. First, connect to the EC2 instance. From there, you can then connect to the RDS instance.

Listing 11.1 Extract from the CloudFormation template setting up an RDS database

```
Resources:
  # [...]
  DatabaseSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'awsinaction-db-sg'
      VpcId: !Ref VPC
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 3306
        ToPort: 3306
        SourceSecurityGroupId: !Ref WebServerSecurityGroup
  Database:
    Type: 'AWS::RDS::DBInstance'
    DeletionPolicy: Delete
    Properties:
      AllocatedStorage: 5
      BackupRetentionPeriod: 0
      DBInstanceClass: 'db.t2.micro'
      DBName: wordpress
      Engine: MySQL
      MasterUsername: wordpress
      MasterUserPassword: wordpress
      VPCSecurityGroups:
      - !Sub ${DatabaseSecurityGroup.GroupId}
      DBSubnetGroupName: !Ref DBSubnetGroup
    DependsOn: VPCGatewayAttachment
  DBSubnetGroup:
    Type: 'AWS::RDS::DBSubnetGroup'
    Properties:
      Description: DB subnet group
      SubnetIds:
      - Ref: SubnetA
      - Ref: SubnetB
```

Security group for the database instance, allowing incoming traffic on the MySQL default port for web servers

The default MySQL port is 3306.

References the security group of EC2 instances running a web server

Creates a database instance with Amazon RDS

Disable backups (turn this to Snapshot in production!).

The database provides 5 GB of storage.

Disable backups (turn this on in production!).

Creates a default database named wordpress

The size of the database instance is t2.micro, the smallest available size.

Uses MySQL as the database engine

The password for the master user of the MySQL database

The username for the master user of the MySQL database

Defines the subnets the RDS database instance will launch into

References the security group for the database instance

Creates a subnet group to define the subnets for the database instance

Uses subnet A or B to launch RDS database instances

See if the CloudFormation stack named `wordpress` has reached the state `CREATE_COMPLETE` with the following command:

```
$ aws cloudformation describe-stacks --stack-name wordpress
```

Search for `StackStatus` in the output, and check whether the status is `CREATE_COMPLETE`. If not, you need to wait a few minutes longer (it can take up to 15 minutes to create the stack) and rerun the command. If the status is `CREATE_COMPLETE`, you'll find the key `OutputKey` in the output section. The corresponding `OutputValue` contains the URL for the WordPress blogging platform. The following listing shows the output in detail. Open this URL in your browser; you'll find a running WordPress setup.

Listing 11.2 Checking the state of the CloudFormation stack

```
$ aws cloudformation describe-stacks --stack-name wordpress
{
  "Stacks": [{
    "StackId": "[...]",
    "Description": "AWS in Action: chapter 11",
    "Parameters": [...],
    "Tags": [],
    "Outputs": [
        {
            "Description": "Wordpress URL",
            "OutputKey": "URL",
            "OutputValue": "http://[...].us-east-1.elb.amazonaws.com"
        }
    ],
    "CreationTime": "2017-10-19T07:12:28.694Z",
    "StackName": "wordpress",
    "NotificationARNs": [],
    "StackStatus": "CREATE_COMPLETE",
    "DisableRollback": false
  }]
}
```

Open this URL in your browser to open the WordPress application.

Wait for state CREATE_COMPLETE for the CloudFormation stack.

Launching and operating a relational database like MySQL is that simple. Of course, you can also use the Management Console (https://console.aws.amazon.com/rds/) to launch an RDS database instance instead of using a CloudFormation template. RDS is a managed service, and AWS handles most of the tasks necessary to operate your database in a secure and reliable way. You only need to do two things:

- Monitor your database's available storage and make sure you increase the allocated storage as needed.
- Monitor your database's performance and make sure you increase I/O and computing performance as needed.

Both tasks can be handled with the help of CloudWatch monitoring, as you'll learn later in the chapter.

11.1.2 Exploring an RDS database instance with a MySQL engine

The CloudFormation stack created an RDS database instance with a MySQL engine. Each instance offers an endpoint for SQL requests. Applications can send their SQL

requests to this endpoint to query, insert, delete, or update data. For example, to retrieve all rows from a table, the application sends the following SQL request: `SELECT * FROM table`. You can request the endpoint and detailed information of an RDS database instance with a `describe` command:

```
$ aws rds describe-db-instances --query "DBInstances[0].Endpoint"
{
  "HostedZoneId": "Z2R2ITUGPM61AM",
  "Port": 3306,
  "Address": "wdwcoq2o8digyr.cqrxioeaavmf.us-east-1.rds.amazonaws.com"
}
```

Port number of database endpoint

Host name of database endpoint

The RDS database is now running, but what does it cost?

11.1.3 Pricing for Amazon RDS

Databases on Amazon RDS are priced according to the size of the underlying virtual machine and the amount and type of allocated storage. Compared to a database running on a plain EC2 VM, the hourly price of an RDS instance is higher. In our opinion, the Amazon RDS service is worth the extra charge because you don't need to perform typical DBA tasks like installation, patching, upgrades, migration, backups, and recovery.

Table 11.3 shows a pricing example for a medium-sized RDS database instance without failover for high availability. All prices in USD are for US East (N. Virginia) as of Nov. 8, 2017. Get the current prices at https://aws.amazon.com/rds/pricing/.

Table 11.3 Monthly (30.5 days) cost for a medium-sized RDS instance

Description	Monthly price
Database instance db.m3.medium	$65.88 USD
50 GB of general purpose (SSD)	$5.75 USD
Additional storage for database snapshots (100 GB)	$9.50 USD
Total	$81.13 USD

You've now launched an RDS database instance for use with a WordPress web application. You'll learn about importing data to the RDS database in the next section.

11.2 Importing data into a database

A database without data isn't useful. In many cases, you'll need to import data into a new database, by importing a dump from the old database for example. If you move your locally hosted systems to AWS, you'll need to transfer the database as well. This section will guide you through the process of importing a MySQL database dump to an RDS database with a MySQL engine. The process is similar for all other database

engines (Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server).

To import a database from your local environment to Amazon RDS, follow these steps:

1. Export the local database.
2. Start a virtual machine in the same region and VPC as the RDS database.
3. Upload the database dump to the virtual machine.
4. Run an import of the database dump to the RDS database on the virtual server.

We'll skip the first step of exporting a MySQL database, because the RDS instance we created in our example is empty and you may not have access to an existing WordPress database. This sidebar gives you hints if you do.

Exporting a MySQL database

MySQL (and every other database system) offers a way to export and import databases. We recommend the command-line tools from MySQL for exporting and importing databases. You may need to install the MySQL client, which comes with the mysqldump tool.

The following command exports all databases from localhost and dumps them into a file called dump.sql. Replace *$UserName* with the MySQL admin or master user, and enter the password when prompted:

```
$ mysqldump -u $UserName -p --all-databases > dump.sql
```

You can also specify only some databases for the export. To do so, replace *$DatabaseName* with the name of the database you want to export:

```
$ mysqldump -u $UserName -p $DatabaseName > dump.sql
```

And of course you can export a database over a network connection. To connect to a server to export a database, replace *$Host* with the host name or IP address of your database:

```
$ mysqldump -u $UserName -p $DatabaseName --host $Host > dump.sql
```

See the MySQL documentation if you need more informations about the `mysqldump` tool.

Theoretically, you could import a database to RDS from any machine in your on-premises or local network. But the higher latency over the internet or VPN connection will slow down the import process dramatically. Because of this, we recommend adding a second step: upload the database dump to a virtual machine running in the same AWS region and VPC, and start to import the database to RDS from there.

> **AWS Database Migration Service**
> When migrating a huge database to AWS with minimal downtime, the Database Migration Service (DMS) can help. We do not cover DMS in this book, but you can learn more on the AWS website: https://aws.amazon.com/dms/.

To do so, we'll guide you through the following steps:

1. Get the public IP address of the virtual machine you want to upload to. We'll be using the VM running WordPress that you created earlier.
2. Connect to the virtual machine via SSH.
3. Download a database dump from S3 to the VM (or if you have an existing database dump that you're migrating, upload it to the VM).
4. Run an import of the database dump to the RDS database from the virtual machine.

Fortunately, you already started two virtual machines that you know can connect to the MySQL database on RDS, because they're running the WordPress application. To find out the public IP address of one of these two virtual machines, run the following command on your local machine:

```
$ aws ec2 describe-instances --filters "Name=tag-key,\
➥ Values=aws:cloudformation:stack-name Name=tag-value,\
➥ Values=wordpress" --output text \
➥ --query "Reservations[0].Instances[0].PublicIpAddress"
```

Open an SSH connection to the VM using the public IP address from the previous command. Use the SSH key `mykey` to authenticate, and replace *`$PublicIpAddress`* with the VM's IP address:

```
$ ssh -i $PathToKey/mykey.pem ec2-user@$PublicIpAddress
```

We prepared a MySQL database dump of a WordPress blog as an example. The dump contains a blog post and a few comments. Download this database dump from S3 using the following command on the virtual machine:

```
$ wget https://s3.amazonaws.com/awsinaction-code2/chapter11/wordpress-import.sql
```

Now you're ready to import the MySQL database dump to the RDS database instance. You'll need the port and hostname, also called the *endpoint*, of the MySQL database on RDS to do so. Don't remember the endpoint? The following command will print it out for you. Run this on your local machine:

```
$ aws rds describe-db-instances --query "DBInstances[0].Endpoint"
```

Run the following command on the VM to import the data from the file wordpress-import.sql into the RDS database instance; replace *$DBHostName* with the RDS endpoint you printed to the terminal with the previous command. Type in the password `wordpress` when asked for a password:

```
$ mysql --host $DBHostName --user wordpress -p < wordpress-import.sql
```

Point your browser to the WordPress blog again, and you'll now find many new posts and comments there. If you don't remember the URL, run the following command on your local machine to fetch it again:

```
$ aws cloudformation describe-stacks --stack-name wordpress \
➥ --query "Stacks[0].Outputs[0].OutputValue" --output text
```

11.3 Backing up and restoring your database

Amazon RDS is a managed service, but you still need backups of your database in case something or someone harms your data and you need to restore it, or you need to duplicate a database in the same or another region. RDS offers manual and automated snapshots for recovering RDS database instances.

In this section, you'll learn how to use RDS snapshots:

- Configuring the retention period and time frame for automated snapshots
- Creating snapshots manually
- Restoring snapshots by starting new database instances based on a snapshot
- Copying a snapshot to another region for disaster recovery or relocation

11.3.1 Configuring automated snapshots

The RDS database you started in section 11.1 can automatically create snapshots if the `BackupRetentionPeriod` is set to a value between `1` and `35`. This value indicates how many days the snapshot will be retained (default is `1`). Automated snapshots are created once a day during the specified time frame. If no time frame is specified, RDS picks a random 30-minute time frame during the night. (A new random time frame will be chosen each night.)

Creating a snapshot requires all disk activity to be briefly frozen. Requests to the database may be delayed or even fail because of a time out, so we recommend that you choose a time frame for the snapshot that has the least impact on applications and users (for example, late at night). Automated snapshots are your backup in case something unexpected happens to your database. This could be a query that deletes all your data accidentally or a hardware failure that causes data loss.

The following command changes the time frame for automated backups to 05:00–06:00 UTC and the retention period to three days. Use the terminal on your local machine to execute it:

```
$ aws cloudformation update-stack --stack-name wordpress --template-url \
➥ https://s3.amazonaws.com/awsinaction-code2/chapter11/ \
➥ template-snapshot.yaml \
➥ --parameters ParameterKey=KeyName,UsePreviousValue=true \
➥ ParameterKey=AdminPassword,UsePreviousValue=true \
➥ ParameterKey=AdminEMail,UsePreviousValue=true
```

The RDS database will be modified based on a slightly modified CloudFormation template, as shown next.

Listing 11.3 Modifying an RDS database's snapshot time frame and retention time

```
Database:
  Type: 'AWS::RDS::DBInstance'
  DeletionPolicy: Delete
  Properties:
    AllocatedStorage: 5
    BackupRetentionPeriod: 3
    PreferredBackupWindow: '05:00-06:00'
    DBInstanceClass: 'db.t2.micro'
    DBName: wordpress
    Engine: MySQL
    MasterUsername: wordpress
    MasterUserPassword: wordpress
    VPCSecurityGroups:
    - !Sub ${DatabaseSecurityGroup.GroupId}
    DBSubnetGroupName: !Ref DBSubnetGroup
  DependsOn: VPCGatewayAttachment
```

Keep snapshots for 3 days.

Create snapshots automatically between 05:00 and 06:00 UTC.

If you want to disable automated snapshots, you need to set the retention period to `0`. As usual, you can configure automated backups using CloudFormation templates, the Management Console, or SDKs. Keep in mind that automated snapshots are deleted when the RDS database instance is deleted. Manual snapshots stay. You'll learn about them next.

11.3.2 Creating snapshots manually

You can trigger manual snapshots whenever you need, for example before you release a new version of your software, migrate a schema, or perform some other activity that could damage your database. To create a snapshot, you have to know the instance identifier. The following command extracts the instance identifier from the first RDS database instance:

```
$ aws rds describe-db-instances   --output text \
➥ --query "DBInstances[0].DBInstanceIdentifier"
```

The following command creates a manual snapshot called `wordpress-manual-snapshot`. Replace *`$DBInstanceIdentifier`* with the output of the previous command.

```
$ aws rds create-db-snapshot --db-snapshot-identifier \
➥ wordpress-manual-snapshot \
➥ --db-instance-identifier $DBInstanceIdentifier
```

It will take a few minutes for the snapshot to be created. You can check the current state of the snapshot with this command:

```
$ aws rds describe-db-snapshots \
➥ --db-snapshot-identifier wordpress-manual-snapshot
```

RDS doesn't delete manual snapshots automatically; you need to delete them yourself if you don't need them any longer. You'll learn how to do this at the end of the section.

Copying an automated snapshot as a manual snapshot

There is a difference between automated and manual snapshots. Automated snapshots are deleted automatically after the retention period is over, but manual snapshots aren't. If you want to keep an automated snapshot even after the retention period is over, you have to copy the automated snapshot to a new manual snapshot.

Get the snapshot identifier of an automated snapshot from the RDS database you started in section 11.1 by running the following command at your local terminal. Replace *$DBInstanceIdentifier* with the output of the `describe-db-instances` command.

```
$ aws rds describe-db-snapshots --snapshot-type automated \
➥ --db-instance-identifier $DBInstanceIdentifier \
➥ --query "DBSnapshots[0].DBSnapshotIdentifier" \
➥ --output text
```

The following command copies an automated snapshot to a manual snapshot named `wordpress-copy-snapshot`. Replace *$SnapshotId* with the output from the previous command:

```
$ aws rds copy-db-snapshot \
➥ --source-db-snapshot-identifier $SnapshotId  \
➥ --target-db-snapshot-identifier wordpress-copy-snapshot
```

The copy of the automated snapshot is named `wordpress-copy-snapshot`. It won't be removed automatically.

11.3.3 Restoring a database

If you restore a database from an automated or manual snapshot, a new database will be created based on the snapshot. As figure 11.2 shows, you can't restore a snapshot to an existing database.

A new database is created when you restore a database snapshot, as figure 11.3 illustrates.

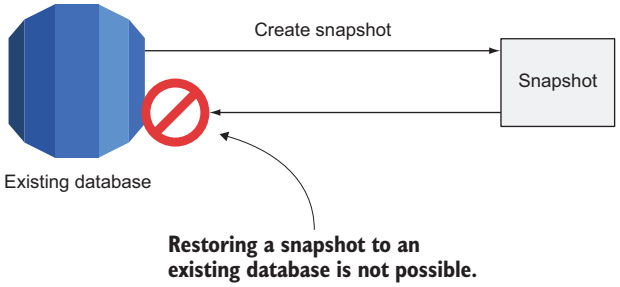

Figure 11.2 A snapshot can't be restored into an existing database.

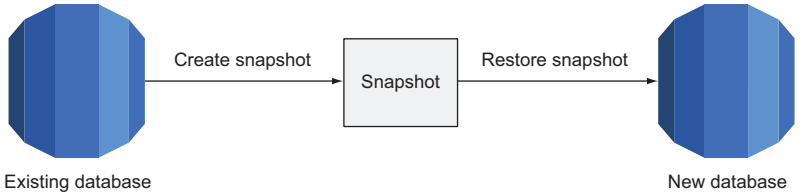

Figure 11.3 A new database is created to restore a snapshot.

To create a new database in the same VPC as the WordPress platform you started in section 11.1, you need to find out the existing database's subnet group. Execute this command to do so:

```
$ aws cloudformation describe-stack-resource \
➥ --stack-name wordpress --logical-resource-id DBSubnetGroup \
➥ --query "StackResourceDetail.PhysicalResourceId" --output text
```

You're now ready to create a new database based on the manual snapshot you created at the beginning of this section. Execute the following command, replacing *$Subnet-Group* with the output of the previous command:

```
$ aws rds restore-db-instance-from-db-snapshot \
➥ --db-instance-identifier awsinaction-db-restore \
➥ --db-snapshot-identifier wordpress-manual-snapshot \
➥ --db-subnet-group-name $SubnetGroup
```

A new database named `awsinaction-db-restore` is created based on the manual snapshot. After the database is created, you can switch the WordPress application to the new endpoint.

If you're using automated snapshots, you can also restore your database from a specified moment, because RDS keeps the database's change logs. This allows you to

jump back to any point in time from the backup retention period to the last five minutes.

Execute the following command, replacing *$DBInstanceIdentifier* with the output of the earlier `describe-db-instances` command, *$SubnetGroup* with the output of the earlier `describe-stack-resource` command, and *$Time* with a UTC timestamp from 5 minutes ago (for example, `2017-10-19T10:55:00Z`):

```
$ aws rds restore-db-instance-to-point-in-time \
➥ --target-db-instance-identifier awsinaction-db-restore-time \
➥ --source-db-instance-identifier $DBInstanceIdentifier \
➥ --restore-time $Time --db-subnet-group-name $SubnetGroup
```

A new database named `awsinaction-db-restore-time` is created based on the source database from 5 minutes ago. After the database is created, you can switch the WordPress application to the new endpoint.

11.3.4 Copying a database to another region

Copying a database to another region is easy with the help of snapshots. The main reasons you might do so are:

- *Disaster recovery*—You can recover from an unlikely region-wide outage.
- *Relocating*—You can move your infrastructure to another region so you can serve your customers with lower latency.

You can easily copy a snapshot to another region. The following command copies the snapshot named `wordpress-manual-snapshot` from the region `us-east-1` to the region `eu-west-1`. You need to replace *$AccountId* with your account ID.

> **COMPLIANCE** Moving data from one region to another may violate privacy laws or compliance rules, especially if the data crosses frontiers. Make sure you're allowed to copy the data to another region if you're working with real data.

```
$ aws rds copy-db-snapshot --source-db-snapshot-identifier \
➥ arn:aws:rds:us-east-1:$AccountId:snapshot:\
➥ wordpress-manual-snapshot --target-db-snapshot-identifier \
➥ wordpress-manual-snapshot --region eu-west-1
```

If you can't remember your account ID, you can look it up with the help of the CLI:

```
$ aws iam get-user --query "User.Arn" --output text
arn:aws:iam::878533158213:user/mycli
```

Account ID has 12 digits (878533158213).

After the snapshot has been copied to the region `eu-west-1`, you can restore a database from it as described in the previous section.

11.3.5 Calculating the cost of snapshots

Snapshots are billed based on the storage they use. You can store snapshots up to the size of your database instance for free. In our WordPress example, you can store up to 5 GB of snapshots for free. On top of that, you pay per GB per month of used storage. As we're writing this book, the cost is $0.095 for each GB every month (region `us-east-1`).

Cleaning up

It's time to clean up the snapshots and delete the restored database instances. Execute the following commands step-by-step, or jump to the shortcuts for Linux and macOS after the listing:

```
$ aws rds delete-db-instance --db-instance-identifier \
➥ awsinaction-db-restore --skip-final-snapshot
$ aws rds delete-db-instance --db-instance-identifier \
➥ awsinaction-db-restore-time --skip-final-snapshot
$ aws rds delete-db-snapshot --db-snapshot-identifier \
➥ wordpress-manual-snapshot
$ aws rds delete-db-snapshot --db-snapshot-identifier \
➥ wordpress-copy-snapshot
$ aws --region eu-west-1 rds delete-db-snapshot --db-snapshot-identifier \
➥ wordpress-manual-snapshot
```

Deletes the database with data from the snapshot restore

Deletes the database with data from the point-in-time restore

Deletes the manual snapshot

Deletes the copied snapshot

Deletes the snapshot copied to another region

You can avoid typing these commands manually at your terminal by using the following command to download a Bash script and execute it directly on your local machine. The Bash script contains the same steps as shown in the previous snippet.

```
$ curl -s https://raw.githubusercontent.com/AWSinAction/\
➥ code2/master/chapter11/cleanup.sh | bash -ex
```

Keep the rest of the setup, because you'll use it in the following sections.

11.4 Controlling access to a database

The shared-responsibility model applies to the RDS service as well as to AWS services in general. AWS is responsible for security of the cloud in this case—for example, for the security of the underlying OS. You, the customer, need to specify the rules controlling access to your data and RDS database.

Figure 11.4 shows the three layers that control access to an RDS database:

1. Controlling access to the configuration of the RDS database
2. Controlling network access to the RDS database
3. Controlling data access with the database's own user and access management features

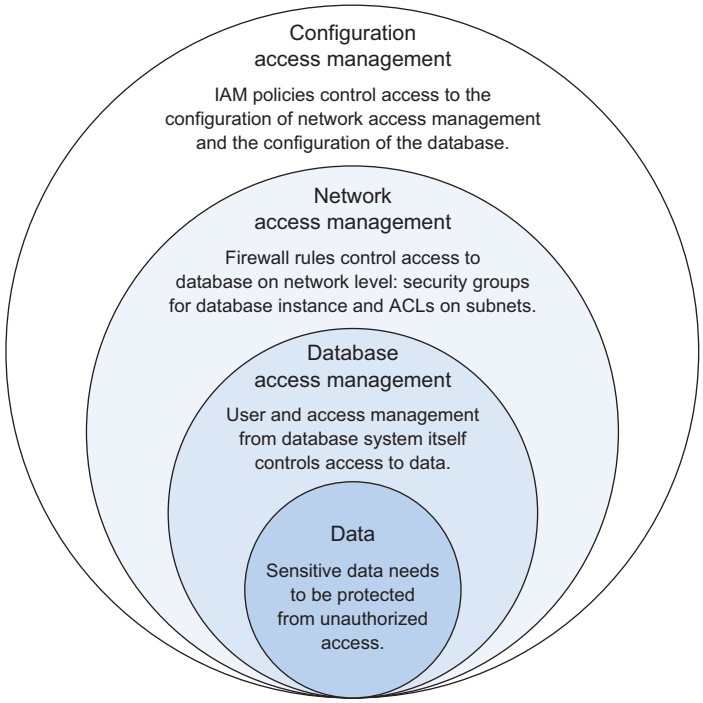

Figure 11.4 Your data is protected by the database itself, security groups, and IAM.

11.4.1 Controlling access to the configuration of an RDS database

Access to the RDS service is controlled using the IAM service. IAM is responsible for controlling access to actions like creating, updating, and deleting an RDS database instance. IAM doesn't manage access inside the database; that's the job of the database engine. IAM policies define which configuration and management actions an identity is allowed to execute on RDS. You attach these policies to IAM users, groups, or roles to control what actions they can perform on the database.

The following listing shows an IAM policy that allows access to all RDS configuration and management actions. You could use this policy to limit access by only attaching it to trusted IAM users and groups.\

Listing 11.4 Allowing access to all RDS service configuration and management actions

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "Stmt1433661637000",
    "Effect": "Allow",
    "Action": "rds:*",
    "Resource": "*"
  }]
}
```

Allows the specified actions on the specified resources

All possible actions on RDS service are specified (for example, changes to the database configuration).

All RDS databases are specified.

Only people and machines that really need to make changes to RDS databases should be allowed to do so. The following listing shows an IAM policy that denies all destructive actions in order to prevent data loss by human failure.

Listing 11.5 IAM policy denying destructive actions

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "Stmt1433661637000",
    "Effect": "Deny",
    "Action": ["rds:Delete*", "rds:Remove*"],
    "Resource": "*"
  }]
}
```

Denies the specified actions on the specified resources

All destructive actions on the RDS service are specified (e.g., delete database instance).

All RDS databases are specified.

See chapter 6 if you're interested in more details about the IAM service.

11.4.2 Controlling network access to an RDS database

An RDS database is linked to security groups. Each security group consists of rules for a firewall controlling inbound and outbound database traffic. You already know about using security groups in combination with virtual machines.

The next listing shows the configuration of the security group attached to the RDS database in our WordPress example. Inbound connections to port 3306 (the default port for MySQL) are only allowed from virtual machines linked to the security group called `WebServerSecurityGroup`.

Listing 11.6 CloudFormation template extract: firewall rules for an RDS database

```
DatabaseSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'awsinaction-db-sg'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - IpProtocol: tcp
      FromPort: 3306
      ToPort: 3306
      SourceSecurityGroupId: !Ref WebServerSecurityGroup
```

Security group for the database instance, allowing incoming traffic on the MySQL default port for web servers

The default MySQL port is 3306.

References the security group for web servers

Only machines that really need to connect to the RDS database should be allowed to do so on the network level, such as EC2 instances running your web server or application server. See chapter 6 if you're interested in more details about security groups (firewall rules).

11.4.3 Controlling data access

A database engine also implements access control itself. User management of the database engine has nothing to do with IAM users and access rights; it's only responsible for controlling access to the database. For example, you typically define a user for each application and grant rights to access and manipulate tables as needed. In the WordPress example, a database user called wordpress is created. The WordPress application authenticates itself to the database engine (MySQL in this case) with this database user and a password.

> **IAM Database Authentication**
>
> AWS has started integrating IAM and the native database authentication mechanism for two engines: MySQL and Aurora. With IAM Database Authentication, you no longer need to create users with a username and password in the database engine. Instead, you create a database user that uses a plugin called `AWSAuthenticationPlugin` for authentication. You then log in to the database with the username and a token that is generated with your IAM identity. The token is valid for 15 minutes, so you have to renew it from time to time. You can learn more about IAM Database Authentication in the AWS documentation at http://mng.bz/5q65.

Typical use cases are as follows:

- Limiting write access to a few database users (for example, only for an application)
- Limiting access to specific tables to a few users (for example, to one department in the organization)
- Limiting access to tables to isolate different applications (for example, hosting multiple applications for different customers on the same database)

User and access management varies between database systems. We don't cover this topic in this book; refer to your database system's documentation for details.

11.5 Relying on a highly available database

The database is typically the most important part of a system. Applications won't work if they can't connect to the database, and the data stored in the database is mission-critical, so the database must be highly available and store data durably.

Amazon RDS lets you launch highly available (HA) databases. Compared to a default database consisting of a single database instance, an HA RDS database consists of two database instances: a master and a standby database. You also pay for both instances. All clients send requests to the master database. Data is replicated between the master and the standby database synchronously, as shown in figure 11.5.

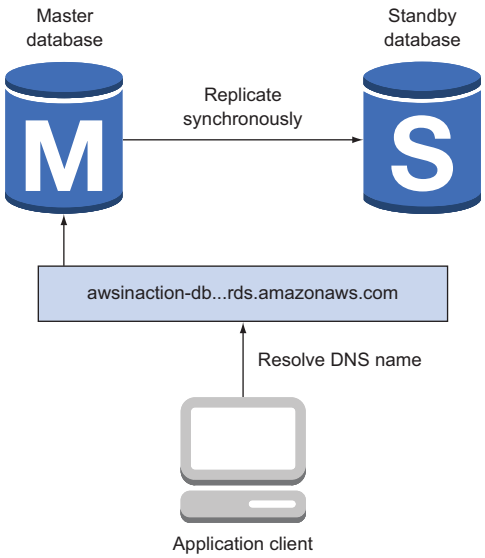

Figure 11.5 The master database is replicated to the standby database when running in high-availability mode.

We strongly recommend using high-availability deployment for all databases that handle production workloads. If you want to save money, you can turn the HA feature off for your test systems.

If the master database becomes unavailable due to hardware or network failures, RDS starts the failover process. The standby database then becomes the master database. As figure 11.6 shows, the DNS name is updated and clients begin to use the former standby database for their requests.

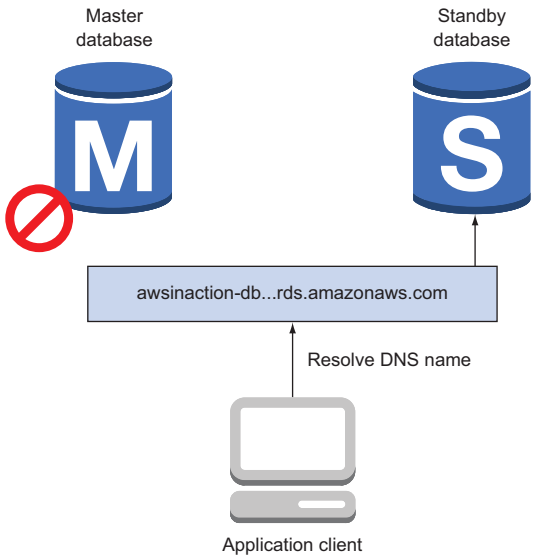

Figure 11.6 The client fails over to the standby database if the master database fails, using DNS resolution.

RDS detects the need for a failover automatically and executes it without human intervention.

> **Aurora is different**
>
> Aurora is an exception. It does not store your data on a single EBS volume. Instead, Aurora stores data on a *cluster volume*. A cluster volume consists of multiple disks, with each disk having a copy of the cluster data. This implies that the storage layer of Aurora is not a single point of failure. But still, only the primary Aurora database instance accepts write requests. If the primary goes down, it is automatically re-created, which typically takes less than 10 minutes. If you have replica instances in your Aurora cluster, a replica is promoted to be the new primary instance, which usually takes around 1 minute and is much faster than primary re-creation.

11.5.1 Enabling high-availability deployment for an RDS database

Execute the following command at your local terminal to enable high-availability deployment for the RDS database you started in section 11.1:

```
$ aws cloudformation update-stack --stack-name wordpress --template-url \
➥ https://s3.amazonaws.com/awsinaction-code2/chapter11/template-multiaz.yaml \
➥ --parameters ParameterKey=KeyName,UsePreviousValue=true \
➥ ParameterKey=AdminPassword,UsePreviousValue=true \
➥ ParameterKey=AdminEMail,UsePreviousValue=true
```

> **WARNING** Starting a highly available RDS database will incur charges. See https://aws.amazon.com/rds/pricing/ if you want to find out the current hourly price.

The RDS database is updated based on a slightly modified CloudFormation template.

Listing 11.7 Modifying the RDS database by enabling high availability

```
Database:
  Type: 'AWS::RDS::DBInstance'
  DeletionPolicy: Delete
  Properties:
    AllocatedStorage: 5
    BackupRetentionPeriod: 3
    PreferredBackupWindow: '05:00-06:00'
    DBInstanceClass: 'db.t2.micro'
    DBName: wordpress
    Engine: MySQL
    MasterUsername: wordpress
    MasterUserPassword: wordpress
    VPCSecurityGroups:
    - !Sub ${DatabaseSecurityGroup.GroupId}
    DBSubnetGroupName: !Ref DBSubnetGroup
    MultiAZ: true
  DependsOn: VPCGatewayAttachment
```

Enables high-availability deployment for the RDS database

It will take several minutes for the database to be deployed in HA mode. But there is nothing more you need to do—the database is now highly available.

> **What is Multi-AZ ?**
> Each AWS region is split into multiple independent data centers, also called *availability zones*. We introduced the concept of availability zones in chapters 9 and 10, but skipped one aspect of HA deployment that is only used for RDS: the master and standby databases are launched into two different availability zones. AWS calls the high-availability deployment of RDS *Multi-AZ* deployment for this reason.

In addition to the fact that a high-availability deployment increases your database's reliability, there is another important advantage. Reconfiguring or maintaining a single-mode database causes short downtimes. High-availability deployment of an RDS database solves this problem because you can switch to the standby database during maintenance.

11.6 Tweaking database performance

The easiest way to scale a RDS database, or a SQL database in general, is to scale *vertically*. Scaling a database vertically means increasing the resources of your database instance:

- Faster CPU
- More memory
- Faster storage

Keep in mind that you can't scale vertically (which means increasing resources) without limits. One of the largest RDS database instance types comes with 32 cores and 244 GiB memory. In comparison, an object store like S3 or a NoSQL database like DynamoDB can be scaled horizontally without limits, as they add more machines to the cluster if additional resources are needed.

11.6.1 Increasing database resources

When you start an RDS database, you choose an instance type. The instance type defines the computing power and memory of your virtual machine (as when you start an EC2 instance). Choosing a bigger instance type increases computing power and memory for RDS databases.

You started an RDS database with instance type `db.t2.micro`, the smallest available instance type. You can change the instance type using a CloudFormation template, the CLI, the Management Console, or AWS SDKs. You may want to increase the instance type if performance is not good enough for you. You will learn how to measure performance in section 11.7. Listing 11.8 shows how to change the CloudFormation template to increase the instance type from `db.t2.micro` with 1 virtual core and 615 MB memory to `db.m3.large` with 2 faster virtual cores and 7.5 GB memory. You'll

do this only in theory. Don't do this to your running database because it is not covered by the Free Tier and will incur charges.

Listing 11.8 Modifying the instance type to improve performance of an RDS database

```
Database:
  Type: 'AWS::RDS::DBInstance'
  DeletionPolicy: Delete
  Properties:
    AllocatedStorage: 5
    BackupRetentionPeriod: 3
    PreferredBackupWindow: '05:00-06:00'
    DBInstanceClass: 'db.m3.large'
    DBName: wordpress
    Engine: MySQL
    MasterUsername: wordpress
    MasterUserPassword: wordpress
    VPCSecurityGroups:
    - !Sub ${DatabaseSecurityGroup.GroupId}
    DBSubnetGroupName: !Ref DBSubnetGroup
    MultiAZ: true
  DependsOn: VPCGatewayAttachment
```

Increases the size of the underlying virtual machine for the database instance from db.t2.micro to db.m3.large

Because a database has to read and write data to a disk, I/O performance is important for the database's overall performance. RDS offers three different types of storage, as you already know from reading about the block storage service EBS:

1. General purpose (SSD)
2. Provisioned IOPS (SSD)
3. Magnetic

You should choose general purpose (SSD) or even provisioned IOPS (SSD) storage for production workloads. The options are exactly the same as when using EBS for virtual machines. If you need to guarantee a high level of read or write throughput, you should use provisioned IOPS (SSD). The general purpose (SSD) option offers moderate baseline performance with the ability to burst. The throughput for general purpose (SSD) depends on the amount of initialized storage size. Magnetic storage is an option if you need to store data at a low cost, or if you don't need to access it in a predictable, performant way. The next listing shows how to enable general purpose (SSD) storage using a CloudFormation template.

Listing 11.9 Modifying the storage type to improve performance of an RDS database

```
Database:
  Type: 'AWS::RDS::DBInstance'
  DeletionPolicy: Delete
  Properties:
    AllocatedStorage: 5
    BackupRetentionPeriod: 3
    PreferredBackupWindow: '05:00-06:00'
```

```
      DBInstanceClass: 'db.m3.large'
      DBName: wordpress
      Engine: MySQL
      MasterUsername: wordpress
      MasterUserPassword: wordpress
      VPCSecurityGroups:
      - !Sub ${DatabaseSecurityGroup.GroupId}
      DBSubnetGroupName: !Ref DBSubnetGroup
      MultiAZ: true
      StorageType: 'gp2'                    <-- Uses general purpose (SSD) storage
    DependsOn: VPCGatewayAttachment             to increase I/O performance
```

11.6.2 Using read replication to increase read performance

A database suffering from too many read requests can be scaled horizontally by adding additional database instances for read replication. As figure 11.7 shows, changes to the database are asynchronously replicated to an additional read-only database instance. The read requests can be distributed between the master database and its read-replication databases to increase read throughput.

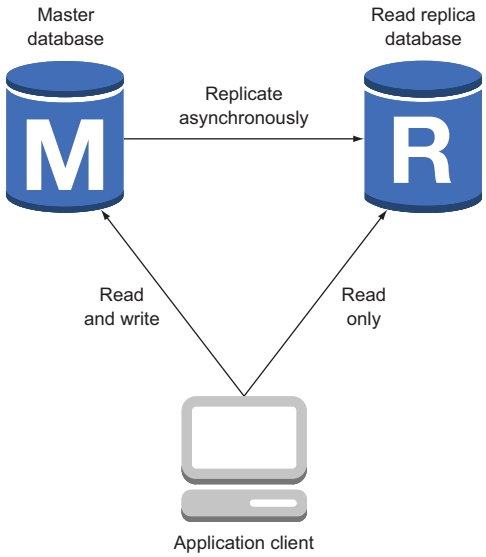

Figure 11.7 Read requests are distributed between the master and read-replication databases for higher read performance.

Tweaking read performance with replication makes sense only if the application generates many read requests and few write requests. Fortunately, most applications read more than they write.

CREATING A READ-REPLICATION DATABASE

Amazon RDS supports read replication for MySQL, MariaDB, and PostgreSQL databases. To use read replication, you need to enable automatic backups for your database, as shown in section 11.3.

WARNING Starting an RDS read replica will incur charges. See https://aws.amazon.com/rds/pricing/ if you want to find out the current hourly price.

Execute the following command from your local machine to create a read-replication database for the WordPress database you started in section 11.1. Replace the *$DBInstanceIdentifier* with the value from `aws rds describe-db-instances --query "DBInstances[0].DBInstanceIdentifier" --output text`.

```
$ aws rds create-db-instance-read-replica \
➥ --db-instance-identifier awsinaction-db-read \
➥ --source-db-instance-identifier $DBInstanceIdentifier
```

RDS automatically triggers the following steps in the background:

1. Creating a snapshot from the source database, also called the master database
2. Launching a new database based on that snapshot
3. Activating replication between the master and read-replication databases
4. Creating an endpoint for SQL read requests to the read-replication database

After the read-replication database is successfully created, it's available to answer SQL read requests. The application using the SQL database must support the use of read-replication databases. WordPress, for example, doesn't support read replicas by default, but you can use a plugin called HyperDB to do so; the configuration is tricky, so we'll skip this part. You can get more information here: https://wordpress.org/plugins/hyperdb/. Creating or deleting a read replica doesn't affect the availability of the master database.

Using read replication to transfer data to another region

RDS supports read replication between regions for Aurora, MySQL, MariaDB, and PostgreSQL databases. You can replicate your data from the data centers in North Virginia to the data centers in Ireland, for example. There are three major use cases for this feature:

1. Backing up data to another region for the unlikely event of an outage covering a complete region
2. Transferring data to another region to be able to answer read requests with lower latency
3. Migrating a database to another region

Creating read replication between two regions incurs an additional cost because you have to pay for the transferred data.

Promoting a read replica to a standalone database

If you create a read-replication database to migrate a database from one region to another, or if you have to perform heavy and load-intensive tasks on your database, such as adding an index, it's helpful to switch your workload from the master database

to a read-replication database. The read replica must become the new master database. Promoting read-replication databases to become master databases is possible for Aurora, MySQL, MariaDB, and PostgreSQL databases with RDS.

The following command promotes the read-replication database you created in this section to a standalone master database. Note that the read-replication database will perform a restart and be unavailable for a few minutes:

```
$ aws rds promote-read-replica --db-instance-identifier awsinaction-db-read
```

The RDS database instance named `awsinaction-db-read` will accept write requests after the transformation is successful.

Cleaning up

It's time to clean up, to avoid unwanted expense. Execute the following command:

```
$ aws rds delete-db-instance --db-instance-identifier \
➥ awsinaction-db-read --skip-final-snapshot
```

You've gained experience with the AWS relational database service in this chapter. We'll end the chapter by taking a closer look at the monitoring capabilities of RDS.

11.7 Monitoring a database

RDS is a managed service. Nevertheless, you need to monitor some metrics yourself to make sure your database can respond to all requests from applications. RDS publishes several metrics for free to AWS CloudWatch, a monitoring service for the AWS cloud. You can watch these metrics through the Management Console, as shown in figure 11.8, and define alarms for when a metric reaches a threshold.

Table 11.4 shows the most important metrics; we recommend that you keep an eye on them by creating alarms.

Table 11.4 Important metrics for RDS databases from CloudWatch

Name	Description
`FreeStorageSpace`	Available storage in bytes. Make sure you don't run out of storage space. We recommend setting the alarm threshold to < 2147483648 (2 GB)
`CPUUtilization`	The usage of the CPU as a percentage. High utilization can be an indicator of a bottleneck due to insufficient CPU performance. We recommend setting the alarm threshold to > 80%.
`FreeableMemory`	Free memory in bytes. Running out of memory can cause performance problems. We recommend setting the alarm threshold to < 67108864 (64 MB).

Table 11.4 Important metrics for RDS databases from CloudWatch *(continued)*

Name	Description
`DiskQueueDepth`	Number of outstanding requests to the disk. A long queue indicates that the database has reached the storage's maximum I/O performance. We recommend setting the alarm threshold to > 64.
`SwapUsage`	If the database has insufficient memory, the OS starts to use the disk as memory (this is called *swapping*). Using the disk as memory is slow and will cause performance issues. We recommend setting the alarm threshold to > 268435456 (256 MB).

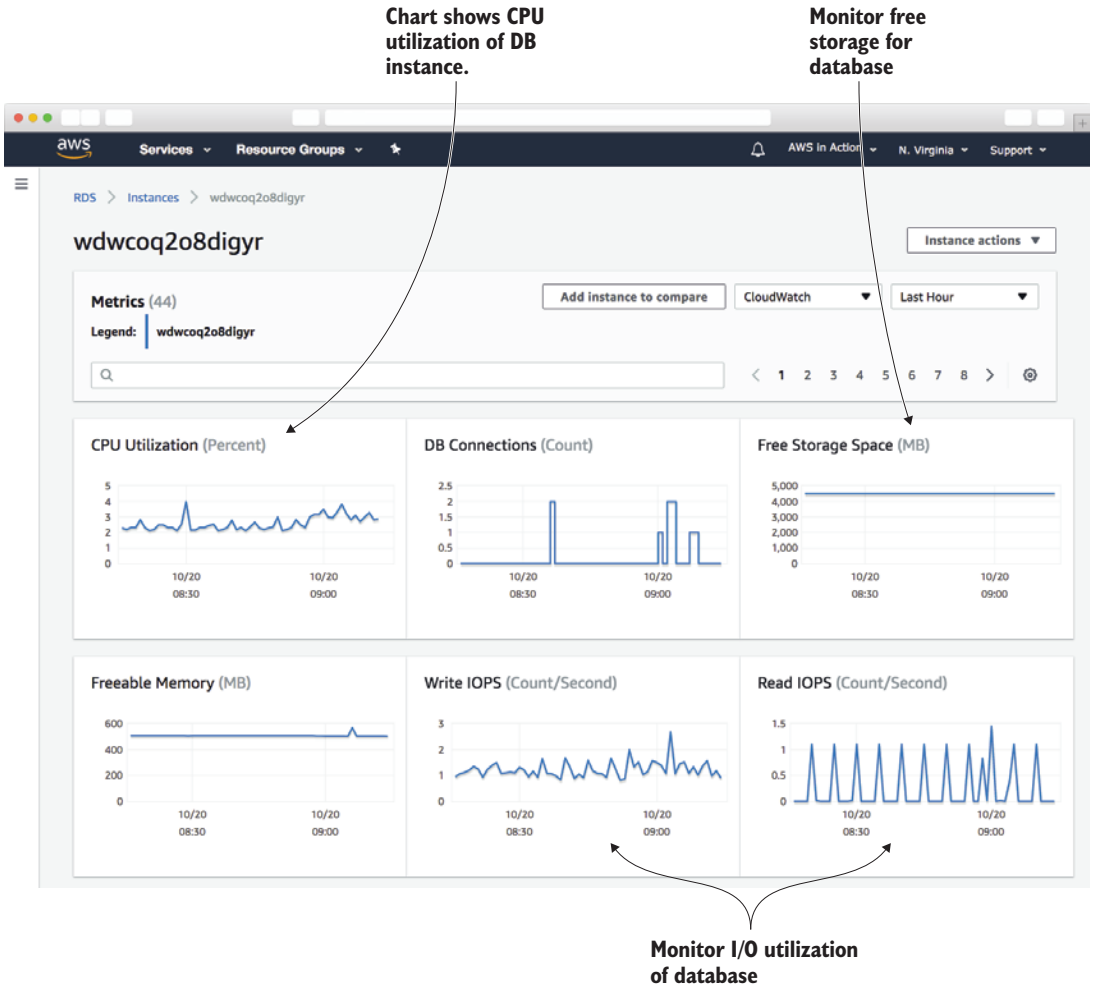

Figure 11.8 Metrics to monitor an RDS database from the Management Console

We recommend that you monitor these metrics in particular, to make sure your database isn't the cause of application performance problems.

Cleaning up

It's time to clean up, to avoid unwanted expense. Execute the following command to delete all resources corresponding to the WordPress blogging platform based on an RDS database:

```
$ aws cloudformation delete-stack --stack-name wordpress
```

In this chapter, you've learned how to use the RDS service to manage relational databases for your applications. The next chapter will focus on a NoSQL database.

Summary

- RDS is a managed service that provides relational databases.
- You can choose between PostgreSQL, MySQL, MariaDB, Oracle Database, and Microsoft SQL Server databases. Aurora is the database engine built by Amazon.
- The fastest way to import data into an RDS database is to copy it to a virtual machine in the same region and pump it into the RDS database from there.
- You can control access to data with a combination of IAM policies and firewall rules, and on the database level.
- You can restore an RDS database to any time in the retention period (a maximum of 35 days).
- RDS databases can be highly available. You should launch RDS databases in Multi-AZ mode for production workloads.
- Read replication can improve the performance of read-intensive workloads on a SQL database.